P.I. Robert Wilson
Laboratory for Atmospheric and Space Physics, University of Colorado Boulder, Boulder, Colorado, USA
Email: Rob.Wilson@lasp.colorado.edu

(Sorry this is rushed and less polished than I'd like, but realized the deadline was this afternoon so I got typing before it was too late…)

In response to topic #1 ('What tools, resources, workflows, tutorials, and interfaces will future users expect or require?')

I have used PDS data a lot in the past, and also peer reviewed some PDS datasets.

The biggest issue to address for me is the ability to read PDS compliant files in to different programing languages easily, to be able to retrieve the files automatically and to confirm that the files are described properly.

While having generic code provided by the PDS to read in any and all PDS data in all common languages (c, Fortran, Python, IDL, Matlab, Mathematica, etc.) on all platforms (PC, Mac, Linux, Solaris, etc.) is near impossible, there should be an easy way to validate your downloaded PDS file. While there is a checksum included in the PDS label files that can be used, that doesn't confirm that the file is obeying the format specified in its own label (or format) file.

The label files contain the number of bytes per record, and what type of data is expected. However in my PDS experience, not all files were as expected (admittedly the vast majority were perfect, but the very few unexpected ones lead to unexpected results, some of which took me hours to identify). When a VALID_MINIMUM and VALID_MAXIMUM were provided, I found instances where the values were outside that range. In text files I've found random characters on odd lines (like an 'a' character when a record should only be numbers, or even a random singular opening parentheses). I've also found instances where a column should be a 2 digit number, but it's 8 digits instead, making the line length different, or the letters NaN or such instead of a number. To easily ready in text data, most codes (Matlab/IDL) expect each line to be the exact same format or length of characters.

There should be an easy way to validate the file obeys the rules from their own label files. Confirming all numbers are between the VALID_MIN and VALID_MAX (or MISSING_CONSTANT) should be an easy check for both binary and text PDS files.

For PDS text files it should be easy to confirm all lines are the same character length and have expected values (numbers or letters, and the correct number of them,

without any erroneous characters, correct number of columns per line, etc.) and nothing else.

While label files do often include a MD5 checksum of the data file, that only tells you if it downloaded correctly and not if the file contents themselves are correct. Similarly, for PDS peer review purposes, reviewers often spot-check data, thus can easily miss the few bad lines in a year's worth of data that would otherwise trip up automatic reading codes and cost you hours in debugging.

For text files this could be simply achieved with a standard one-line (likely one very long line) of regular expression that could be used to validate the files. Regular Expression is a bit of an art form, there's a billion ways to describe everything – and often it's clear to the creator as they compose it but is hard for others to read, but it should not be difficult to make one regular expression phrase that checks for correct character length and type. Once tested and included within the label file, no one needs to read the regular expression to understand it – they just can trust it works and copy/paste it to then apply it.

When I peer reviewed the Cassini Magnetometer data I did this, and knocked up this regular expression line to look at the 1-second resolution data:
/^\d{4}-(0[1-9]|1[012])-(0[1-9]|[12]\d|3[01])T([01]\d|2[0123]):[0-5]\d:([0-5]\d|60)\.\d{1}(\s((\d{5})|([\-\s]\d{4})|(\s[\-\s]\d{3})|(\s{2}[\-\s]\d{2})|(\s{3}[\-\s]\d{1}))\.\d{3}){3}(\s((\d{4})|(\s\d{3})|(\s{2}\d{2})|(\s{3}\d{1}))\.\d{3}){1}\s((\d\d)|(\s\d))\r?\n?$/

This was far better than spot checking data as I could run it on every single record in the volume, but is not fully comprehensive – it could be better. e.g. while the above started by checking for a UTC data string of the form yyyy-mm-ddTHH:MM:SS where mm could only be 01-12, dd as 01-31, it would still allow bad dates such as Feb 31st. It forces hours to be 00-23 only, minutes to be 0-59 only and seconds to be 0-60 (to allow for a leap second), but again did not check that if the whole second was 60 that it occurred at 23:59:60 at the end of June or December only (when leap seconds can occur), and doesn't allow for the possibility of adding 2 leap seconds in one go – allowed but as yet never occurred (23:59:61).

While checking a UTC sting sounds complex, someone only needs to create a regular expression check for it once for both yyyy-mm-dd and yyyy-ddd formats, and that same regular expression string can be used for all files.

Likewise the last column is described as ((\d\d)|(\s\d)), which lets either a 2 digits number or a single digit number pre-pended with a space through, which would have caught a recent error I found and reported (and was fixed in PDS quickly) where a 2-character number had the value "-2147483648" by mistake in the file.

I admit this regular expression approach looks horrendous – but it's really not – and would give end users an instant way to confirm the PDS data files (when text) are in the expected format.

Wish #1 for Future of PDS: Regular Expression

My first wish for the future of PDS text files would to include a regular expression phrase of a valid record within the label file to allow file contents to be automatically verified by anyone.

The PDS could do this as a matter of course when they receive new files to the volume, but it is also good to give the end user the ability to check the file syntax.

Wish #2 for Future of PDS: Check data against valid ranges

My second wish is that binary files have an easy way (perhaps in c or python, or another free language that anyone can download with no pay-wall) to confirm that a whole number of records exist and each objects entry is within the valid min/max range as specified by the label file.

Wish #3 for Future of PDS: Have a way to pull down files automatically

My third wish is that by knowing PDS volume name and file name one could create a web request (via HTTPS) to retrieve that file locally. As such end users codes could be written to automatically get data when it needs it, as it needs it, rather than relying on the user to download data themselves prior to anlaysis.

Wish #4 for Future of PDS : The perfect file format for all… (impossible)

One file format for all possible data… but practical impossible, as cited by XKCD and https://xkcd.com/927/. However some thoughts follow on the remaining pages.

The hunt for a perfect data format – thoughts for consideration

My fourth wish is to have the perfect PDS file suitable for all data… sadly that is as likely a team of unicorns winning the Superb Owl (spelling deliberate), there is no one perfect file format for the PDS as people's data are too different.

While I would love there to be just one file type – the best I can do is to explain what I think is less ideal in the current types of file on the PDS.

Text files are easy to read as a human, and will be over the decades – a great plus for archiving standards, however there are downsides. The first is that they can be huge in file size, and are not well suited to multidimensional data. They do not email well either; personally I've hit issues where someone emails me a PDS file they've been working on and the email programs used alter the line endings, affecting record lengths. We've moved to gzipping PDS text files before emailing them to each other to preserve line endings.

Comma separated variable (CSV) text files seem ideal (especially when they don't have to be fixed length records) and are read my many programs, however they are very 'English'. The French, Italians and the main international auxiliary languages (such as Esperanto) use commas for a decimal point, resulting in CSV files becoming very confusing. A tab delimited file would be better as a world wide language standard. Does the PDS have a specific language? Must all files be submitted in English? For that matter and aiming at a US audience, would Spanish language files also be allowed?

Although going out of fashion, I like PDS3 binary files. The perceived downside is they require some coding in order to read the files, however the label files clearly state byte type (integer, float, etc.), size, etc. have no issues with line endings so can be emailed cleanly, but again have issues with large dimensional data (e.g. 3D+). These are clearly labeled, and any computer programmer should have zero trouble decoding them – although many scientists seem to have an illogical fear of this, it really is very straightforward. I think the PDS got this right for long term archiving – does the job perfectly – but the PDS could use more examples of how to deal with 3D or even 4D and above data in this format.

CDF files seem to be the new vogue, however I believe them to be currently unsuitable for PDS. They require 3rd party software to read (from Goddard, the built in CDF routines in IDL and Matlab are too old and not compatible with current CDF versions), which also gets updated every so often. Does Goddard have a requirement to keep CDF reading software available for as long as PDS files are expected to be around? For instance the Goddard Space Physics Data Facility have already stopped supporting CDF software for CDF versions 2.5, 2.6 and 2.7 (the latter being from 2012) on 3 different hardware platforms due to 'lack of [user] interest'. While that makes sense for day to day use, it's terrible for archiving purposes when data must be accessible for decades to come on whatever platforms

people may have. Who's to say their software for the current CDF version 3.0 will still be accessible in a decade, let alone 100 years from now?

My personal opinion of CDFs being unsuitable for the PDS however is based on the current format, and may be fixable. A huge benefit of CDFs is the ability to store multi-dimensional data, however there are draw backs for an archival stand-alone product:

1) The description fields within the CDF usually have an 80 character limit, which is just too short. For some of my descriptions fields in PDS 3 files I run to many pages to give a clear description or to simply note the 8 different values this object could take and what each means. For a new CDF version I wish the description fields did not have any limit on size.
2) Without downloading and using the Goddard CDF codes these files are difficult to treat as a simple binary file. I know for the MAVEN mission the PDS folks have worked on a label file that could be used to decode the CDF, but this seems far more complex that a regular flat-file PDS3 version binary.
3) A UTC time string must be an allowed EPOCH that other CDF variables can reference from (or DEPEND on in CDF speak) without using TT2000. UTC time strings are the PDS preferred method of time, for very good reasons, so the CDF should natively support that.
4) CDF files are not stand alone files! They require the use of a leap second file to convert between their (nano-)seconds since an epoch and a human yyyy-mm-dd HH:MM:SS time (or yyyy-ddd) that most scientists work in. Currently the Goddard CDF software downloads contain a leap second file stored on your local machine – but a user has no immediate way to check if that is the most current leap second file or not. As such, all data is not stand alone, and requires use of a secondary file that has no easy way to keep updated. And if I were to share a CDF file with a colleague, I've no way to know what leap second file they have (perhaps from an earlier Goddard software release), meaning we could see different 'human' times for the same data.

The latter two are the most critical aspects as to why I think the PDS should not use CDF files in their current form. The CDF teams have recently made a way to include reference leap second arrays within the same CDF per file, which is good, but unless the Goddard CDF software (and any inbuilt CDF software in Matlab/IDL) always uses the leap second array within the same CDF instead of any other local file there will be room for confusion. I'd also like to see the Goddard CDF team join forces with the NAIF SPICE teams to use their leap second kernel so that the NASA family has just one leap second file of reference.

These are fixable problems for later versions of CDF formats if the will is there, but for the foreseeable future I will not personally create CDF files as an acceptable PDS format for low level (CODMAC 2) spacecraft data I work on, as that is the basis for all higher level products so must be correct without any doubts whatsoever.